Programma del corso

- *Introduzione agli algoritmi*
- ***Rappresentazione delle Informazioni***
- *Architettura del calcolatore*
- *Reti di Calcolatori (Reti Locali, Internet)*
- *Elementi di Programmazione*

Addizione di Numeri al Complemento di 2

- Tramite l'addizione normale a base di 2!
- Con N bit, eventuali N+1-esimi bit nel risultati sono scartati
- Il segno viene determinato “automaticamente”.
- Esempio: 15 + –5 (utilizzando 8 bit)

```
 1 1 1 1 1   1 1 1      (riporto)
  0000 1111       ⇨ 15₁₀
  1111 1011       ⇨ -5₁₀
 ===========
1 0000 1010       ⇨ 0000 1010 ⇨ 10₁₀
```

8+1-esimo bit viene scartato

Complemento a due: Vantaggi e svantaggi

Vantaggi:

- Addizione “automatica”
- Un solo valore per 0
- Ordine dei numeri mantenuto

Svantaggi:

- Conversione leggermente più complicata

Rappresentazione di numeri frazionari: Virgola fissa

Un numero frazionario è rappresentato come una coppia di numeri interi: la parte intera e la parte decimale.

12,75 scritto come una coppia: <12;0,75>

<1100; 11> ⇨

$1*2^{3}+1*2^{2}+0*2^{1}+0*2^{0}+ 1*2^{-1} + 1*2^{-2}$

Numeri in virgola mobile (**Floating point**)

Idea: $12{,}52 = 1252/100 = 1252 * 10^{-2}$

Un numero decimale è rappresentato come un intero moltiplicato per una opportuna potenza di 10, cioè con una coppia:

<1252; -2>

mantissa esponente

Numeri floating point (binari)

E’ necessario stabilire quanti bit assegnare alla mantissa e all’esponente.

Ad esempio, con 16 bit a disposizione possiamo usarne 12 per la mantissa e 4 per l’esponente

(la mantissa e l’esponente sono di solito espressi in complemento a 2, per cui un bit corrisponde al segno della mantissa e uno a quello dell’esponente)

Numeri floating point (binari)

Con lo stesso metodo possiamo rappresentare numeri molto grandi. Ad esempio, con 8 bit:

<0111;0111>

4 bit di mantissa: $0111_2 = 7_{10}$

4 bit di esponente: $0111_2 = 7_{10}$

<0111;0111> ⇨ $(7 * 2^7)_{10} = 896_{10}$

Mentre, con la notazione classica, con 8 bit rappresentiamo al massimo il numero 255

Numeri floating point

Ma allora, perchè non usare sempre la notazione floating point?

Perchè si perde in precisione

Esempio: 5 cifre (decimali) : 4 per la mantissa, 1 per l'esponente. Rappresentare

312,45

$<3124; -1> = [312,4 .. 312,5]???$

Numeri floating point

Quindi: possiamo rappresentare numeri molto grandi o con molti decimali al costo di una perdita di precisione

Perché? Perché i computer permettono solo rappresentazioni **finite**, e così dobbiamo approssimare alcuni numeri (ad esempio gli irrazionali), ma anche **immagini e suoni**

La Codifica dei Caratteri

AB…ab…&%$…

Codici per i simboli dell'alfabeto

- Per rappresentare i simboli dell'alfabeto anglosassone (0 1 2 … A B … a b …) bastano 7 bit (codifica **ASCII**)
  - Nota: *B* e *b* sono simboli diversi
  - 26 maiuscole + 26 minuscole + 10 cifre + 30 segni di interpunzione+… -> circa 120 oggetti

- Per l'alfabeto esteso con simboli quali à, è, €, … bastano 8 bit nelle codifiche accettate universalmente chiamata **ASCII esteso**
- Per manipolare un numero maggiore di simboli si utilizzano codifiche di **UNICODE**

Codifica ASCII

- La codifica **ASCII** (**A**merican **S**tandard **C**ode for **I**nformation **I**nterchange) utilizza codici su 7 bit **(2^7 = 128 caratteri diversi)**
- Ad esempio
  - 1 0 0 0 0 0 1 rappresenta A
  - 1 0 0 0 0 1 0 rappresenta B
  - 1 0 0 0 0 1 1 rappresenta C
- Le parole si codificano utilizzando sequenze di valori da 7 bit
  - 1000010 1000001 1000010 1000001
    B A B A

Altri codici di codifica

- **ASCII ESTESO**
  - Usa anche il primo bit di ogni byte
  - 256 caratteri diversi
  - non è standard (cambia con la lingua usata)
- **ISO 8859-1: contiene i caratteri latini di maggior uso (coincide con ASCII per i primi 127 valori)**
- **UNICODE (UTF-8 e UTF-16)**
  - standard proposto a 8 e 16 bit (65.536 caratteri)
  - UTF-8 è usato per le e-mail
- **EBCDIC**
  - altro codice a 8 bit della IBM (quasi in disuso)

Tabella ASCII (0-127)

00000000	Null	00100000	Spc	01000000	@	01100000	`
00000001	Start of heading	00100001	!	01000001	A	01100001	a
00000010	Start of text	00100010	"	01000010	B	01100010	b
00000011	End of text	00100011	#	01000011	C	01100011	c
00000100	End of transmit	00100100	$	01000100	D	01100100	d
00000101	Enquiry	00100101	%	01000101	E	01100101	e
00000110	Acknowledge	00100110	&	01000110	F	01100110	f
00000111	Audible bell	00100111	'	01000111	G	01100111	g
00001000	Backspace	00101000	(	01001000	H	01101000	h
00001001	Horizontal tab	00101001	)	01001001	I	01101001	i
00001010	Line feed	00101010	*	01001010	J	01101010	j
00001011	Vertical tab	00101011	+	01001011	K	01101011	k
00001100	Form Feed	00101100	,	01001100	L	01101100	l
00001101	Carriage return	00101101	-	01001101	M	01101101	m
00001110	Shift out	00101110	.	01001110	N	01101110	n
00001111	Shift in	00101111	/	01001111	O	01101111	o
00010000	Data link escape	00110000	0	01010000	P	01110000	p
00010001	Device control 1	00110001	1	01010001	Q	01110001	q
00010010	Device control 2	00110010	2	01010010	R	01110010	r
00010011	Device control 3	00110011	3	01010011	S	01110011	s
00010100	Device control 4	00110100	4	01010100	T	01110100	t
00010101	Neg. acknowledge	00110101	5	01010101	U	01110101	u
00010110	Synchronous idle	00110110	6	01010110	V	01110110	v
00010111	End trans. block	00110111	7	01010111	W	01110111	w
00011000	Cancel	00111000	8	01011000	X	01111000	x
00011001	End of medium	00111001	9	01011001	Y	01111001	y
00011010	Substitution	00111010	:	01011010	Z	01111010	z
00011011	Escape	00111011	;	01011011	[	01111011	{
00011100	File separator	00111100	<	01011100	\	01111100	|
00011101	Group separator	00111101	=	01011101	]	01111101	}
00011110	Record Separator	00111110	>	01011110	^	01111110	~
00011111	Unit separator	00111111	?	01011111	_	01111111	Del

“Numeri” in ASCII

Le cifre 0..9 rappresentate in Ascii sono caratteri e **NON** quantità numeriche

Non possiamo usarle per indicare quantità e per le operazioni aritmetiche. (Anche nella vita di tutti giorni usiamo i numeri come simboli e non come quantità: i n. telefonici)